Introduction to the Theory of Formal Languages

Wiebke Petersen

Heinrich-Heine-Universität Düsseldorf
Institute of Language and Information
Computational Linguistics
www.phil-fak.uni-duesseldorf.de/~petersen/

Riga, 2006

Outline

alphabets and words

Definition

- **alphabet** Σ: nonempty, finite set of **symbols**
- **word**: a finite string $x_1 \ldots x_n$ of symbols.
- **length** of a word $|w|$: number of symbols of a word w (example: $|abbaca| = 6$)
- **empty word** ϵ: the word of length 0
- Σ^* is the set of all words over Σ
- Σ^+ is the set of all nonempty words over Σ ($\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$)

Substrings

Prefix, suffix, infix

Given words $w, v \in \Sigma^*$:

prefix: w is a prefix of v iff there exists a word $u \in \Sigma^*$ with $v = wu$.

suffix: w is a suffix of v iff there exists a word $u \in \Sigma^*$ with $v = uw$.

infix: w is an infix of v iff there exist words $u_1, u_2 \in \Sigma^*$ with $v = u_1 w u_2$.

Concatenation

Definition

The **concatenation** of two words $w = a_1 a_2 \dots a_n$ and $v = b_1 b_2 \dots b_m$ with $n, m \geq 0$ is

$$w \circ v = a_1 \dots a_n b_1 \dots b_m$$

Sometimes we write uv instead of $u \circ v$.

$$w \circ \epsilon = \epsilon \circ w = w \quad \text{neutral element}$$

$$u \circ (v \circ w) = (u \circ v) \circ w \quad \text{associativity}$$

$(\Sigma^*, \circ)$ is a semi-group with neutral element (monoid).

Exponents and reversals

Exponents

- w^n: w concatenated n-times with itself.
- $w^0 = \epsilon$
- $w^* = \bigcup_{n \geq 0} w^n$
- $\epsilon \in w^*$ for any word w

Reversals

- The reversal of a word w is denoted w^R (example: $(abcd)^R = dcba$.
- A word w with $w = w^R$ is called a **palindrome**.

(madam, mum, otto, anna,...)

Formal language

Definition

A **formal language** L is a set of words over an alphabet Σ, i.e. $L \subseteq \Sigma^*$.

Examples:

- language L_{pal} of the palindromes in English
  $L_{pal} = \{\text{mum, madam}, \dots\}$
- language L_{Mors} of the letters of the latin alphabet encoded in the Morse code: $L_{Mors} = \{\cdot-, -\cdot\cdot\cdot, \dots, --\cdot\cdot\}$
- the empty set
- the set of words of length 13 over the alphabet $\{a, b, c\}$
- English?

Operations on formal languages

Definition

- If $L \subseteq \Sigma^*$ and $K \subseteq \Sigma^*$ are two formal languages over an alphabet Σ, then
$$K \cup L,\ K \cap L,\ K \setminus L$$
are languages over Σ too.
- The **concatenation** of two formal languages K and L is
$$K \circ L := \{v \circ w \in \Sigma^* | v \in K, w \in L\}$$
- $L^n = \underbrace{L \circ L \circ L \ldots \circ L}_{n\text{-times}}$
- $L^* := \bigcup_{n \geq 0} L^n$. Note: $\{\epsilon\} \in L^*$ for any language L.

Operations on formal languages

Example

$K = \{abb, a\}$ and $L = \{bbb, ab\}$

- $K \circ L = \{abbbbb, abbab, abbb, aab\}$ and
  $L \circ K = \{bbbabb, bbba, ababb, aba\}$
- $K \circ \emptyset = \emptyset$
- $K \circ \{\epsilon\} = K$
- $K^2 = \{abbabb, abba, aabb, aa\}$

Enumerating all elements of a language

- Peter says that Mary is fallen of the tree.
- Oskar says that Peter says that Mary is fallen of the tree.
- Lisa says that Oskar says that Peter says that Mary is fallen of the tree.
- …

The set of strings of a natural language is infinite.

The enumeration does not gather generalizations.

Grammars

Grammar

- A formal grammar is a **generating device** which can generate (and analyze) strings/words.
- Grammars are finite rule systems.
- The set of all strings generated by a grammar is a formal language (= generated language).

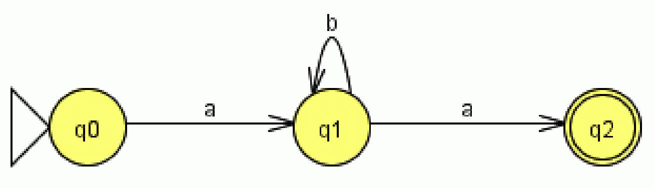
$$\begin{array}{lllllllll} S & \rightarrow & NP\ VP & VP & \rightarrow & V & NP & \rightarrow & D\ N \\ D & \rightarrow & the & N & \rightarrow & cat & V & \rightarrow & sleeps \end{array}$$

Generates: the cat sleeps

Automata

Automaton

- An automaton is a **recognizing device** which accepts strings/words.
- The set of all strings accepted by an automaton is a formal language (=accepted language).



accepts: $L(ab^\star a)$

Formal grammar

Definition

A **formal grammar** is a 4-tupel $G = (N, T, S, P)$ with

- an alphabet of terminals T,
- an alphabet of nonterminals N with $N \cap T = \emptyset$,
- a start symbol $S \in N$,
- a finite set of rules/productions
  $P \subseteq \{\langle \alpha, \beta \rangle \mid \alpha, \beta \in (N \cup T)^* \text{ and } \alpha \notin T^*\}$.

Instead of $\langle \alpha, \beta \rangle$ we write also $\alpha \to \beta$.

Formal grammar

Vocabulary

Let $G = (N, T, S, P)$ be a grammar and $v, w \in (T \cup N)^*$:

- v is directly derived from w (or w directly generates v), $w \to v$ if $w = w_1 \alpha w_2$ and $v = w_1 \beta w_2$ such that $\langle \alpha, \beta \rangle \in P$.
- v is **derived** from w (or w **generates** v), $w \to^* v$ if there exists $w_0, w_1, \ldots w_k \in (T \cup N)^*$ $(k \geq 0)$ such that $w = w_0$, $w_k = v$ and $w_{i-1} \to w_i$ for all $k \geq i \geq 0$.
- $\to^*$ denotes the reflexive transitive closure of $\to$
- $L(G) = \{w \in T^* | S \to^* w\}$ is the formal language generated by the grammar G.

Example

$$G_1 = \langle \{S,NP,VP,N,V,D,N,EN\}, \{the, cat, peter, chases\}, S, P \rangle$$

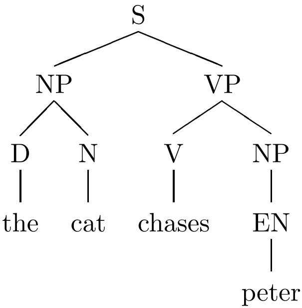
$$P = \left\{ \begin{array}{rclrclrcl} S & \rightarrow & NP\ VP & VP & \rightarrow & V\ NP & NP & \rightarrow & D\ N \\ NP & \rightarrow & EN & D & \rightarrow & the & N & \rightarrow & cat \\ EN & \rightarrow & peter & V & \rightarrow & chases & & & \end{array} \right\}$$

$$L(G_1) = \left\{ \begin{array}{cc} \text{the cat chases peter} & \text{peter chases the cat} \\ \text{peter chases peter} & \text{the cat chases the cat} \end{array} \right\}$$

"the cat chases peter" can be derived from S by:

$$\begin{array}{llll} S & \rightarrow NP\ VP & \rightarrow NP\ V\ NP & \rightarrow NP\ V\ EN \\ & \rightarrow NP\ V\ peter & \rightarrow NP\ chases\ peter & \rightarrow D\ N\ chases\ peter \\ & \rightarrow D\ cat\ chases\ peter & \rightarrow \text{ the cat chases peter} & \end{array}$$

Derivation tree



One derivation determines one derivation tree, but
the same derivation tree can result from different derivations.

Not all formal languages are derivable from a formal grammar

- The set of all formal languages over an alphabet $\Sigma = \{a\}$ is $\mathcal{POW}(\Sigma^*)$; hence, the set is uncountably infinite.
- The set of grammars generating formal languages over Σ with finite sets of productions is countably infinite.
- Hence, the set of formal languages generated by a formal grammar is a strict subset of the set of all formal languages.

Chomsky-hierarchy

- The Chomsky-hierarchy is a hierarchy over structure conditions on the productions.
- Constraining the structure of the productions results in a restricted set of languages.
- The language classes correspond to conditions on the right- and left-hand sides of the productions.
- The Chomsky-hierarchy reflects a special form of complexity, other criteria are possible and result in different hierarchies.
- Linguists benefit from the rule-focussed definition of the Chomsky-hierarchy.

Noam Chomsky

Noam Chomsky
($\ast$ 7.12.1928, Philadelphia)
Noam Chomsky, *Three Models for the Description of Language*, (1956).

Chomsky-hierarchy

A grammar (N, T, S, P) is a

Type 3) **regular grammar (REG)**: iff every production is of the form

$A \rightarrow \beta B$ or $A \rightarrow \beta$ with $A, B \in N$ and $\beta \in T^*$(right-linear grammar);

or iff every production is of the form

$A \rightarrow B\beta$ or $A \rightarrow \beta$ with $A, B \in N$ and $\beta \in T^*$(left-linear grammar).

Type 2) **context-free grammar (CF)**: iff every production is of the form

$$A \rightarrow \beta \text{ with } A \in N \text{ and } \beta \in (N \cup T)^*.$$

Chomsky-hierarchy (cont.)

A grammar (N, T, S, P) is a

Type 1) **context-sensitive grammar (CS)**: iff every production is of the form

$$\gamma A\delta \rightarrow \gamma\beta\delta \text{ with } \gamma, \delta, \beta \in (N \cup T)^*, A \in N \text{ and } \beta \neq \epsilon;$$

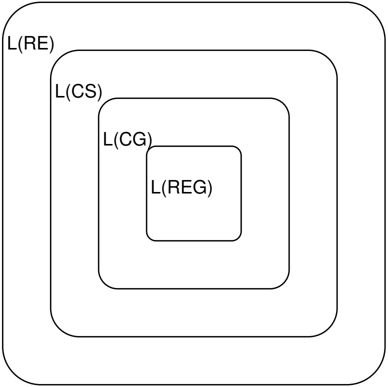
or of the form $S \rightarrow \epsilon$, in which case S does not occur on any right-hand side of a production.

Type 0) **phrase-structure grammar (recursively enumerable grammar) (RE)**: iff every production is of the form

$$\alpha \rightarrow \beta \text{ with } \alpha \in (N \cup T)^* \setminus T^* \text{ and } \beta \in (N \cup T)^*.$$

Main theorem

$$\mathbf{L(REG)} \subset \mathbf{L(CG)} \subset \mathbf{L(CS)} \subset \mathbf{L(RE)}$$



Regular expressions

RE: syntax

The set of **regular expressions** RE_Σ over an alphabet $\Sigma = \{a_1, \ldots, a_n\}$ is defined by:

- $\underline{\emptyset}$ is a regular expression.
- ϵ is a regular expression.
- $a_1, \ldots, a_n$ are regular expressions
- If a and b are regular expressions over Σ then
  - $(a + b)$
  - $(a \bullet b)$
  - $(a^\star)$

  are regular expressions too.

(The brackets are frequently omitted w.r.t. the following dominance scheme: $\star$ dominates $\bullet$ dominates $+$)

Regular expressions

RE: semantics

Each regular expression r over an alphabet Σ denotes a formal language $L(r) \subseteq \Sigma^*$.
Regular languages are those formal languages which can be expressed by a regular expression.
The denotation function L is defined inductively:

- $L(\underline{\emptyset}) = \emptyset$, $L(\epsilon) = \{\epsilon\}$, $L(a_i) = \{a_i\}$
- $L(a + b) = L(a) \cup L(b)$
- $L(a \bullet b) = L(a) \circ L(b)$
- $L(a^\star) = L(a)^*$

Type3-languages

Definition

A grammar (N, T, S, P) is **right-linear** iff all productions are of the form:

$$A \rightarrow w \text{ or } A \rightarrow wB \text{ with } A, B \in N \text{ and } w \in T^*.$$

A language generated by a right-linear grammar is said to be a right-linear language.

Proposition

If L is a formal language, the following statements are equivalent:

1. *L is right-linear*
2. *L is regular*
3. *(L is left-linear)*

Proof: Each regular language is right-linear

$\Sigma = \{a_1, \dots, a_n\}$

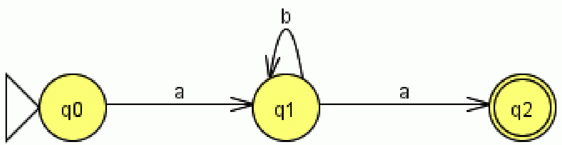
1. $\emptyset$ is generated by $(\{S\}, \Sigma, S, \{\})$,
2. $\{\epsilon\}$ is generated by $(\{S\}, \Sigma, S, \{S \rightarrow \epsilon\})$,
3. $\{a_i\}$ is generated by $(\{S\}, \Sigma, S, \{S \rightarrow a_i\})$,
4. If L_1, L_2 are regular languages with generating right-linear grammars (N_1, T_1, S_1, P_1), (N_2, T_2, S_2, P_2), then $L_1 \cup L_2$ is generated by $(N_1 \uplus N_2, T_1 \cup T_2, S, P_1 \cup_{\uplus} P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\})$,
5. $L_1 \circ L_2$ is generated by $(N_1 \uplus N_2, T_1 \cup T_2, S_1, P_1' \cup_{\uplus} P_2)$ (P_1' is obtained from P_1 if all rules of the form $A \rightarrow w$ ($w \in T^*$) are replaced by $A \rightarrow wS_2$),
6. L_1^* is generated by $(N_1, \Sigma, S_1, P_1' \cup \{S_1 \rightarrow \epsilon\})$ (P_1' is obtained from P_1 if all rules of the form $A \rightarrow w$ ($w \in T^*$) are replaced by $A \rightarrow wS_1$).

Deterministic finite-state automaton (DFSA)

Definition

*A **deterministic finite-state automaton** is a tuple $\langle Q, \Sigma, \delta, q_0, F\rangle$ with:*

1. *a finite, non-empty set of **states** Q*
2. *an alphabet Σ with $Q \cap \Sigma = \emptyset$*
3. *a partial **transition** function $\delta : Q \times \Sigma \rightarrow Q$*
4. *an **initial state** $q_0 \in Q$ and*
5. *a set of **final states** $F \subseteq Q$.*



Language accepted by an automaton

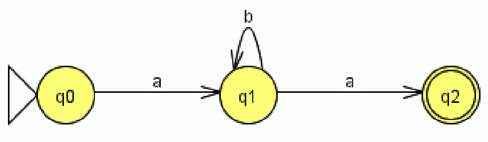
Definition

*A **situation** of a finite-state automaton $\langle Q, \Sigma, \delta, q_0, F\rangle$ is a triple (x, q, y) with $x, y \in \Sigma^*$ and $q \in Q$.*
*Situation (x, q, y) **produces** situation (x', q', y') **in one step** if there exists an $a \in \Sigma$ such that $x' = xa$, $y = ay'$ and $\delta(q, a) = q'$, we write $(x, q, y) \vdash (x', q', y')$ ($(x, q, y) \vdash^* (x', q', y')$ as usual).*

Definition

*A word $w \in \Sigma^*$ gets **accepted** by an automaton $\langle Q, \Sigma, \delta, q_0, F\rangle$ if $(\epsilon, q_0, w) \vdash^* (w, q_n, \epsilon)$ with $q_n \in F$.*
An automaton accepts a language iff it accepts every word of the language.

Example



accepts $L(ab^{\star}a)$

Nondeterministic finite-state automaton NDFSA

Definition

A ***nondeterministic finite-state automaton*** *is a tuple* $\langle Q, \Sigma, \Delta, q_0, F \rangle$ *with:*

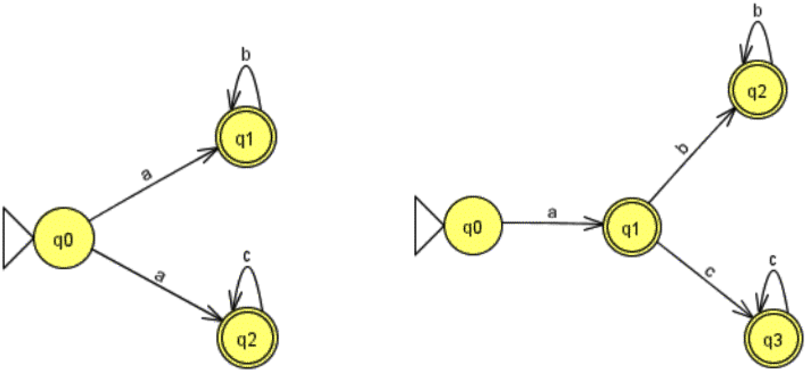
1. *a finite non-empty set of* ***states*** Q
2. *an alphabet* Σ *with* $Q \cap \Sigma = \emptyset$
3. **a transition relation** $\Delta \subseteq Q \times \Sigma \times Q$
4. *an* ***initial state*** $q_0 \in Q$ *and*
5. *a set of* ***final states*** $F \subseteq Q$.

Theorem

A language L can be accepted by a DFSA iff L can be accepted by a NFSA.

Example DEA / NDEA

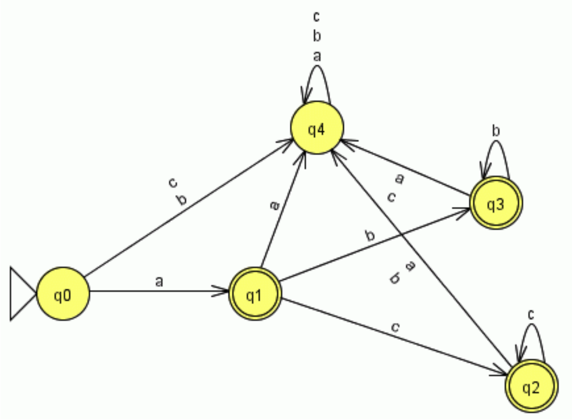
The language $L(ab^{\star} + ac^{\star})$ gets accepted by



Note: Even automatons with ϵ-transitions accept the same languages like NDEA's.

Complete deterministic finite-state automata

Complete deterministic finite-state automata have a total transition function:

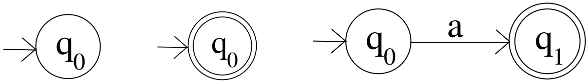


Finite-state automatons accept regular languages

Theorem (Kleene)

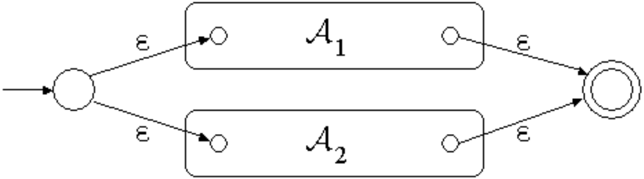
Every language accepted by a DFSA is regular and every regular language is accepted by some DFSA.

proof idea: Each regular language is accepted by a NDFSA:



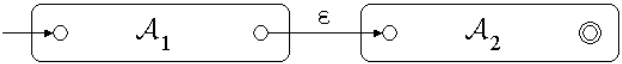
Proof of Kleene's theorem (cont.)

If R_1 and R_2 are two regular expressions such that the languages $L(R_1)$ and $L(R_2)$ are accepted by the automatons $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively, then $L(R_1 + R_2)$ is accepted by:



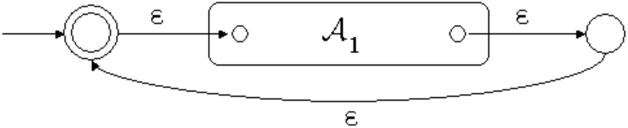
Proof of Kleene's theorem (cont.)

$L(R_1 \bullet R_2)$ is accepted by:



Proof of Kleene's theorem (cont.)

$L(R_1^*)$ is accepted by:



Closure properties of regular languages

	Type3	Type2	Type1	Type0
union	+ ✓	+	+	+
intersection	+	-	+	+
complement	+	-	+	-
concatenation	+ ✓	+	+	+
Kleene's star	+ ✓	+	+	+
intersection with a regular language	+	+	+	+

complement: construct complementary DFSA

intersection: implied by *de Morgan*

Pumping lemma for regular languages

Lemma (Pumping-Lemma)

If L is an infinite regular language over Σ, then there exists words $u, v, w \in \Sigma^*$ such that $v \neq \epsilon$ and $uv^iw \in L$ for any $i \geq 0$.

proof sketch:

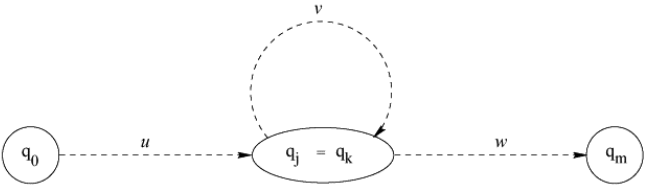
- Any regular language is accepted by a DFSA with a finite number n of states.
- Any infinite language contains a word z which is longer than n ($|z| \geq n$).
- While reading in z, the DFSA passes at least one state q_j twice.

Pumping lemma for regular languages (cont.)

Lemma (Pumping-Lemma)

If L is an infinite regular language over Σ, then there exists words $u, v, w \in \Sigma^*$ *such that* $v \neq \epsilon$ *and* $uv^iw \in L$ *for any* $i \geq 0$.

proof sketch:



$L = \{a^n b^n \ : \ n \geq 0\}$ is not regular

- $L = \{a^n b^n \ : \ n \geq 0\}$ is infinite.
- Suppose L is regular. Then there exists $u, v, w \in \{a, b\}^*$, $v \neq \epsilon$ with $uv^n w \in L$ for any $n \geq 0$.
- We have to consider 3 cases for v.
  1. v consists of a's and b's.
  2. v consists only of a's.
  3. v consists only of b's.

Exercises

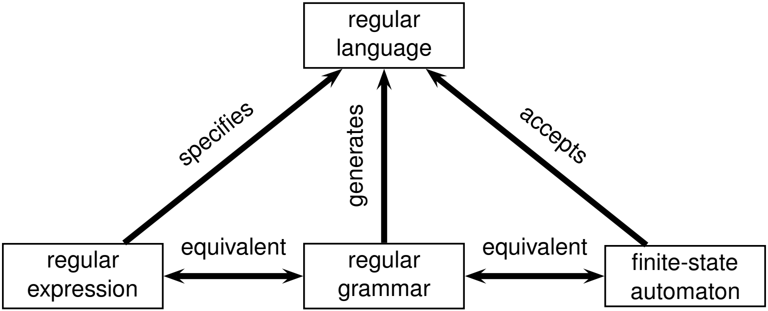
Are the following languages regular?

1. $L_1 = \{w \in \{a, b\}^* \ : \ w \text{ contains an even number of } b's\}$.
2. $L_2 = \{w \in \{a, b\}^* \ : \ w \text{ contains as many } b's \text{ as } a's\}$.
3. $L_3 = \{ww^R \in \{a, b\}^* \ : \ ww^R \text{ is a palindrome over } \{a, b\}^*\}$.

Intuitive rules for regular languages

- L is regular if it is possible to check the membership of a word simply by reading it symbol for symbol while using only a finite stack.
- Finite-state automatons are too weak for:
  - counting in $\mathbb{N}$ ("same number as");
  - recognizing a pattern of arbitrary length ("palindrome");
  - expressions with brackets of arbitrary depth.

Summary: regular languages



Context-free language

Definition

A grammar (N, T, S, P) is ***context-free*** *if all production rules are of the form:*

$$A \rightarrow \alpha, \text{ with } A \in N \text{ and } \alpha \in (T \cup N)^*.$$

A language generated by a context-free grammar is said to be context-free.

Proposition

The set of context-free languages is a strict superset of the set of regular languages.

Proof: Each regular language is per definition context-free. $L(a^n b^n)$ is context-free but not regular ($S \rightarrow aSb$, $S \rightarrow \epsilon$).

Examples of context-free languages

- $L_1 = \{ww^R : w \in \{a,b\}^*\}$
- $L_2 = \{a^i b^j : i \geq j\}$
- $L_3 = \{w \in \{a,b\}^* : \text{more } a's \text{ than } b's\}$
- $L_4 = \{w \in \{a,b\}^* : \text{number of } a's \text{ equals number of } b's\}$

$$\left\{ \begin{array}{lllllllll} S & \rightarrow & aB & A & \rightarrow & a & B & \rightarrow & b \\ S & \rightarrow & bA & A & \rightarrow & aS & B & \rightarrow & bS \\ & & & A & \rightarrow & bAA & B & \rightarrow & aBB \end{array} \right\}$$

Ambiguous grammars and ambiguous languages

Definition

Given a context-free grammar G: A derivation which always replaces the left furthest nonterminal symbol is called ***left-derivation***

Definition

A context-free grammar G is ***ambiguous*** *iff there exists a* $w \in L(G)$ *with more than one left-derivation,* $S \rightarrow^* w$.

Definition

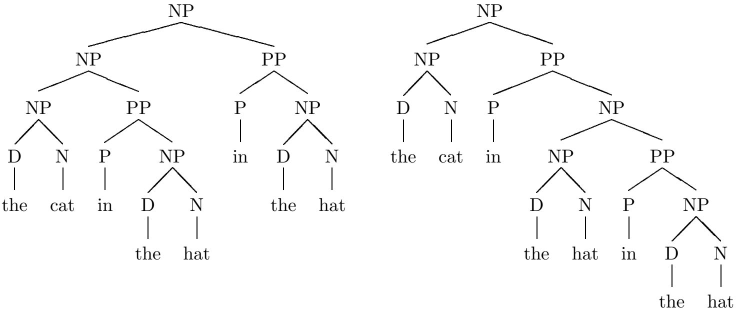
A context-free language L is ***ambiguous*** *iff each context-free grammar G with* $L(G) = L$ *is ambiguous.*

Left-derivations and derivation trees determine each other!

Example of an ambiguous grammar

$G = (N, T, \mathrm{NP}, P)$ with $N = \{\mathrm{D}, \mathrm{N}, \mathrm{P}, \mathrm{NP}, \mathrm{PP}\}$, $T = \{\mathrm{the}, \mathrm{cat}, \mathrm{hat}, \mathrm{in}\}$,

$$P = \left\{ \begin{array}{lllllllll} \mathrm{NP} & \rightarrow & \mathrm{D\ N} & \mathrm{D} & \rightarrow & \mathrm{the} & \mathrm{N} & \rightarrow & \mathrm{hat} \\ \mathrm{NP} & \rightarrow & \mathrm{NP\ PP} & \mathrm{N} & \rightarrow & \mathrm{cat} & \mathrm{P} & \rightarrow & \mathrm{in} \\ \mathrm{PP} & \rightarrow & \mathrm{P\ NP} & & & & & & \end{array} \right\}$$



Chomsky Normal Form

Definition

*A grammar is in **Chomsky Normal Form (CNF)** if all production rules are of the form*

1. $A \rightarrow a$
2. $A \rightarrow BC$

with $A, B, C \in T$ and $a \in \Sigma$ (and if necessary $S \rightarrow \epsilon$ in which case S may not occur in any right-hand side of a rule).

Proposition

Each context-free language is generated by a grammar in CNF.

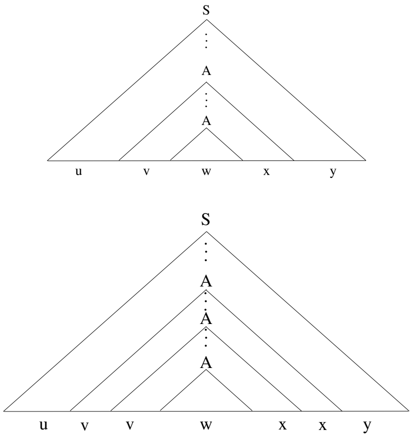
Pumping lemma for context-free languages

Lemma (pumping lemma)

For each context-free language L there exists a $p \in \mathbb{N}$ such that for any $z \in L$: if $|z| > p$, then z may be written as $z = uvwxy$ with

- $u, v, w, x, y \in T^*$,
- $|vwx| \leq p$,
- $vx \neq \epsilon$ and
- $uv^iwx^iy \in L$ for any $i \geq 0$.

Pumping lemma: proof sketch



$|vwx| \leq p$, $vx \neq \epsilon$ and $uv^iwx^iy \in L$ for any $i \geq 0$.

Existence of non context-free languages

- $L_1 = \{a^n b^n c^n\}$
- $L_2 = \{a^n b^m c^n d^m\}$
- $L_1 = \{ww : w \in \{a, b\}^*\}$

Closure properties of context-free languages

	Type3	Type2	Type1	Type0
union	+	+	+	+
intersection	+	-	+	+
complement	+	-	+	-
concatenation	+	+	+	+
Kleene's star	+	+	+	+
intersection with a regular language	+	+	+	+

union: $G = (N_1 \uplus N_2 \cup \{S\}, T_1 \cup T_2, S, P)$ with
$P = P_1 \cup_{\uplus} P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}$

intersection: $L_1 = \{a^n b^n a^k\}$, $L_2 = \{a^n b^k a^k\}$, but $L_1 \cap L_2 = \{a^n b^n a^n\}$

complement: *de Morgan*

concatenation: $G = (N_1 \uplus N_2 \cup \{S\}, T_1 \cup T_2, S, P)$ with
$P = P_1 \cup_{\uplus} P_2 \cup \{S \rightarrow S_1 S_2\}$

Kleene's star: $G = (N_1 \cup \{S\}, T_1, S, P)$ with
$P = P_1 \cup \{S \rightarrow S_1 S, S \rightarrow \epsilon\}$

decision problems

Given: grammars $G = (N, \Sigma, S, P)$, $G' = (N', \Sigma, S', P')$, and a word $w \in \Sigma^*$

word problem Is w derivable from G ?

emptiness problem Does G generate a nonempty language?

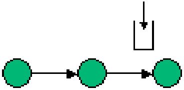
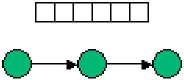
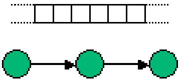
equivalence problem Do G and G' generate the same language ($L(G) = L(G')$)?

Results for the decision problems

	Type3	Type2	Type1	Type0
word problem	D	D	D	U
emptiness problem	D	D	U	U
equivalence problem	D	U	U	U

D: decidable; U: undecidable

Chomsky-hierarchy (1956)

Type 3: REG	finite-state automaton		WP: linear
Type 2: CF	pushdown-automaton		WP: cubic
Type 1: CS	linearly restricted automaton		WP: exponential
Type 0: RE	Turing machine		WP: not decidable

Literature

Beesley & Karttunen (2003) *Finite State Morphology*. CSLI.

Hopcroft, Motwani & Ullman (2001) *Introduction to Automata and Language Theory*. Addison-Wesley, 2nd edition.

Partee, ter Meulen & Wall (1990) *Mathematical Methods in Linguistics*. Kluwer Academic Publishers.

Sipser (2005) *Introduction to the Theory of Computation*. Thomson Course Technology, 2nd edition.

Sudkamp (1996) *Languages and Machines: An Introduction to the Theory of Computer Science*. Addison Wesley, 2nd edition.